目录

[02百度搜索安装Neo4j和JDK 1](#_Toc104925149)

[03CQL基础操作： 1](#_Toc104925150)

[03neo4j︱与python结合的py2neo使用教程 3](#_Toc104925151)

[04 建立基础图数据库 12](#_Toc104925152)

[05 医疗知识图谱问答机器人 13](#_Toc104925153)

# 02百度搜索安装Neo4j和JDK

[Windows安装neo4j图数据库及python.py2neo全流程 - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/436687958)

[JDK+NEO4J安装及配置-----记录 - it610.com](https://www.it610.com/article/1305855080527859712.htm)

先试试是否安装成功了：java -version

启动，在命令行输入:neo4j.bat console

浏览器输入 <http://localhost:7474/> （注意命令行窗口不能关闭）

# 03CQL基础操作：

[neo4j︱图数据库基本概念、操作罗列与整理（一）\_悟乙己的博客-CSDN博客\_图数据库基本概念](https://mattzheng.blog.csdn.net/article/details/79883503)

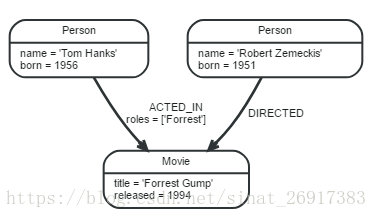
增删改查等基本功能

**neo4j︱图数据库基本概念、操作罗列与整理**

一、neo4j 基本操作元素

neo4j可支持语言：.NET、Java、Spring、JavaScript、Python、Ruby、PHP、R、Go、C / C++、Clojure、Perl、Haskell

几个专有名词：变量（标识符）、节点、关系、实体、标签、属性、索引、约束。



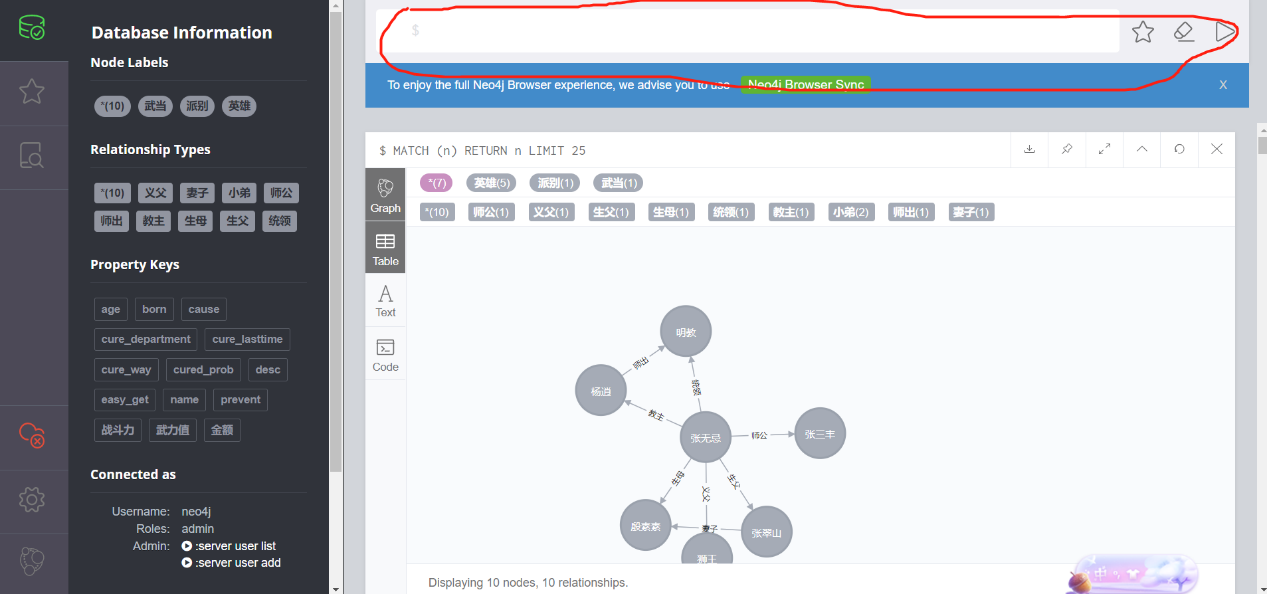
此结构中有：5个实体，三个节点和两个关系，实体包括节点和关系  
实体：Person和Movie  
关系类型：ACTED\_ID和DIRECTED  
节点和关系的属性，name，title，roles等

**变量**

MATCH (n)-->(b) RETURN b  
变量用于引用搜索模式（Pattern），但是变量不是必需的，如果不需要引用，那么可以忽略变量，譬如()就叫匿名变量。  
小括号（）中为命令变量环节，同时其区分大小写

**二、数据库的增删改查**

运行程序前保证命令行是空的，并且运行一下空程序

****

还是按照数据库的一贯风格，增删改查：

Neo4j增删改查：

增：

增加一个节点

create (n:Person {name:'我',age:33}) #有name，age两个属性

带有关系属性

create (p:Person{name:"我",age:"33"})-[:包工程{金额:10000}]->(n:Person{name:"包工头",age:"35"})

create (n:Person{name:"我",age:"33"})-[:包工程{金额:10000}]->(n:Person{name:"包工头",age:"35"}) #这么写会报错Can't create node `n` with labels or properties here. The variable is already declared in this context

删节点

create (n:Person {name:'suijimingzi',age:33})

match (n:Person {name:'suijimingzi'}) delete n

删除关系

match (p:Person{name:"我",age:"33"})-[f:包工程]->(n:Person{name:"包工头",age:"35"}) delete f

改：

加上标签

match (t:Person) where id(t)=1175 set t:好人 return t #id(t)根据实际情况写

加上属性

match (a:好人) where id(a)=789 set a.工作能力=200 return a

修改属性

match (a:好人) where id(a)=789 set a.工作能力=500 return a

查：(查操作太多啦，直接参考neo4j例子就好)，下面的例子不能实现，因为这个关系已经被删掉了

match (p:Person) - [:包工程] -> (n:Person) return p,n

快速清空数据库：

MATCH (n)

DETACH DELETE n

# 03neo4j︱与python结合的py2neo使用教程

[neo4j的python.py2neo操作入门 - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/437824721)使用py2neo实现增删改查经典例子

[The Py2neo Handbook — py2neo 2021.1](https://py2neo.org/2021.1/index.html) 在线官方手册

对工具包进行安装

pip install py2neo -i https://pypi.tuna.tsinghua.edu.cn/simple

**节点/关系/属性/路径**

节点(Node)和关系(relationship)是构成图的基础，节点和关系都可以有多个属性（property），并且均可以作为实体。  
**重点：**

1. 关系必须包含两个节点，且具有方向：start node →end node
2. 对于一个节点来说，与之相连的关系是有输入和输出两个方向。（如node2有输入关系和输出关系：node1→node2→node3），这个特性对于遍历图很重要。
3. 一个节点可以有一个关系是指向自己的。
4. 属性：键-值（key-value），键是字符串类型，值，可以是原数据，也可以由原数据同类型的数组。
5. 路径：查询或者遍历的结果（至少包括一个node）。最短的路径就是一个node，长度为0。

**数据类型**

1. Node：节点，基本语法：Node（\*labels，\*\*properties）（注意：代码中，test\_graph.create(node\_1)的作用是将本地创建的node放入数据库中，后面关系、路径等，在本地创建以后，均需要create)。

from py2neo import Node, Relationship

from py2neo import Graph

test\_graph=Graph("http://localhost:7474", auth=("neo4j", "wangmeng"))

*# 定义node*

node\_1 **=** Node('英雄',name **=** '张无忌')

node\_2 **=** Node('英雄',name **=** '杨逍',武力值**=**'100')

node\_3 **=** Node('派别',name **=** '明教')

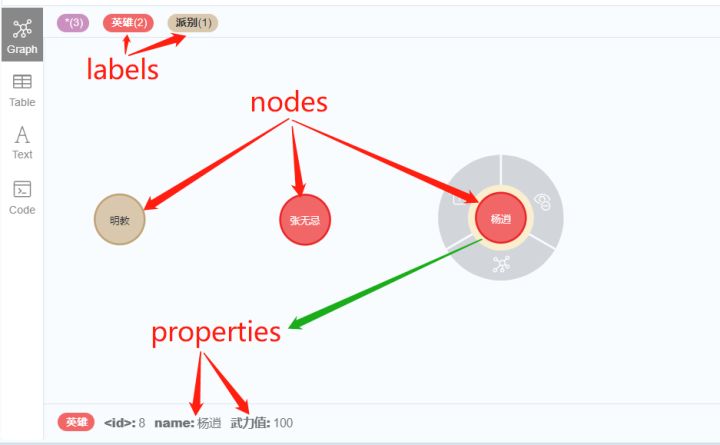
*# 存入图数据库*

test\_graph**.**create(node\_1)

test\_graph**.**create(node\_2)

test\_graph**.**create(node\_3)

print(node\_1)



2. Relationship:关系，基本语法：Relationship（(*start\_node*, *type*, *end\_node*, *\*\*properties*)）

# 增加关系

node\_1\_to\_node\_2 = Relationship(node\_1,'教主',node\_2)

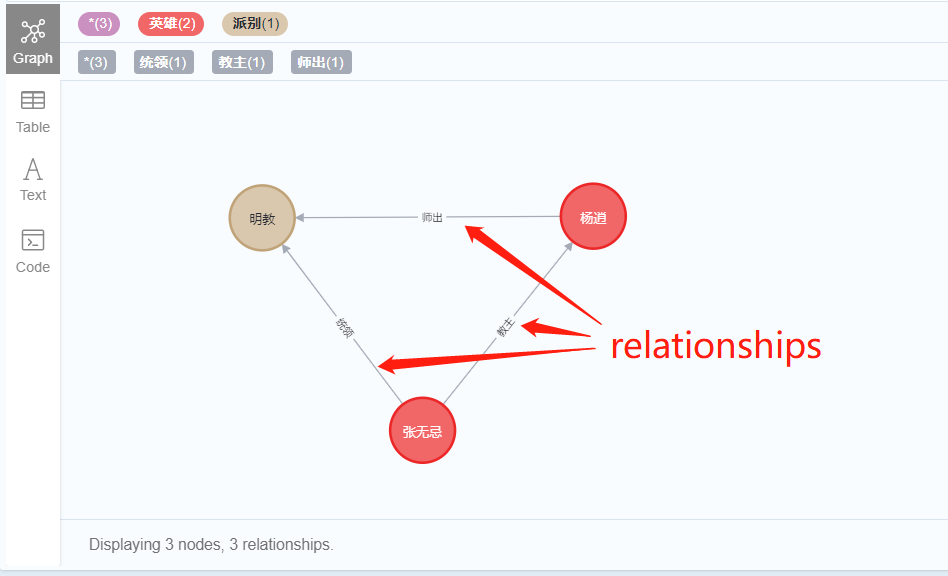
node\_3\_to\_node\_1 = Relationship(node\_1,'统领',node\_3)

node\_2\_to\_node\_2 = Relationship(node\_2,'师出',node\_3)

test\_graph.create(node\_1\_to\_node\_2)

test\_graph.create(node\_3\_to\_node\_1)

test\_graph.create(node\_2\_to\_node\_2)



3. Path:路径，基本语法：Path(\*entities)，注意entities是实体（还记得什么是实体吗，关系，节点都可以作为实体）。

from py2neo import Path

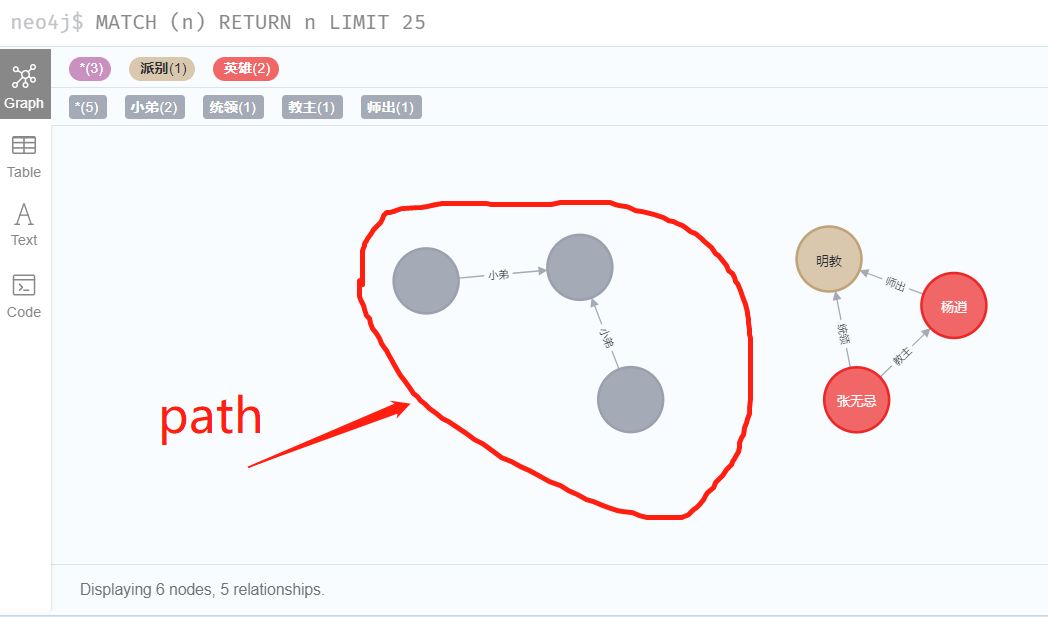
# 建一个路径：比如按照该路径查询，或者遍历的结果保存为路径

node\_4,node\_5,node\_6 = Node(name='阿大'),Node(name='阿二'),Node(name='阿三')

path\_1 = Path(node\_4,'小弟',node\_5,Relationship(node\_6, "小弟", node\_5),node\_6)

test\_graph.create(path\_1)

print(path\_1)



4. Subgraph:子图是节点和关系的任意集合，它也是 Node、Relationship 和 Path 的基类。基本语法：Subgraph(nodes, relationships)。空子图表示为None，使用bool（）可以测试是否为空。参数要按数组输入，如下面代码。

# 创建一个子图，并通过子图的方式更新数据库

node\_7 = Node('英雄',name = '张翠山')

node\_8 = Node('英雄',name = '殷素素')

node\_9 = Node('英雄',name = '狮王')

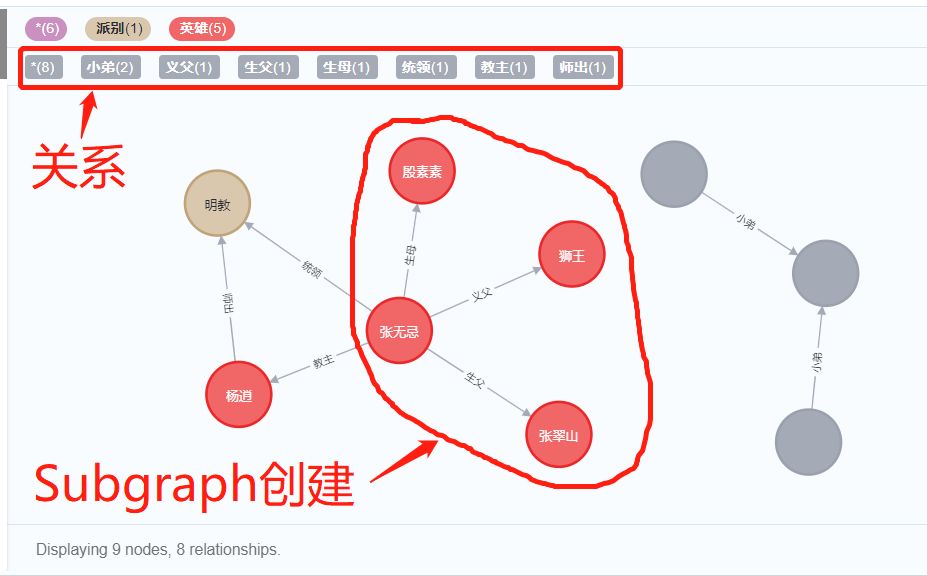
relationship7 = Relationship(node\_1,'生父',node\_7)

relationship8 = Relationship(node\_1,'生母',node\_8)

relationship9 = Relationship(node\_1,'义父',node\_9)

subgraph\_1 = Subgraph(nodes = [node\_7,node\_8,node\_9],relationships = [relationship7,relationship8,relationship9])

test\_graph.create(subgraph\_1)



**工作流**

1. GraphService：基于图服务的工作流。
2. Graph：基于图数据库的工作流（前文所述的基本上都是如此）。
3. Transaction：基于事务的工作流，比较重要。（还记得前面说的AICD吗，我们在一个事务里，进行多种操作，只有全部完成才算工作流完成哦，比如：一个Transaction分两个任务：（1）增加一个新节点（2）将该节点与已有节点创建新关系。这俩任务有一个没完成，整个工作流就不会生效。通常，该种方式通过Graph.begain(readonly=False)构造函数构造，参数readonly表示只读，不给参数默认开始写。（你觉得这个参数重要吗？万一有一天，你想查看别人的数据库，是不是只读比较好呢）。示例代码如下，示例中是写的操作的：

# 创建一个新的事务

transaction\_1 = test\_graph.begin()

# 创建一个新node

node\_10 = Node('武当',name = '张三丰')

transaction\_1.create(node\_10)

# 创建两个关系：张无忌→（师公）→张三丰 张翠山→（妻子）→殷素素

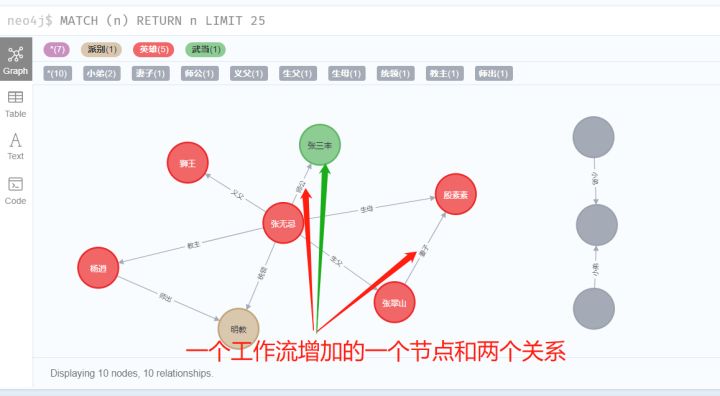
relationship\_10 = Relationship(node\_1,'师公',node\_10)

relationship\_11 = Relationship(node\_7,'妻子',node\_8)

transaction\_1.create(relationship\_10)

transaction\_1.create(relationship\_11)

transaction\_1.commit()



**删**

前面介绍重要概念时，已经将增加节点和关系穿插其中了，批量增加节点的方法是可以通过导入csv或其他文件来执行，内容将在neo4j入门篇最后一篇介绍。下面着重介绍删改查，其中删除节点或者关系方法如下：

1. 删除数据库中所有节点和关系：test\_graph.delete\_all()
2. 其他删除方法如下（删除的基础是查询，但凡查询条件没错，就不会删错）：

# 删除所有：谨慎使用

# test\_graph.delete\_all()

# 按照节点id删除：要删除某个节点之前，需要先删除关系。否则会报错：ClientError

test\_graph.run('match (r) where id(r) = 3 delete r')

# 按照name属性删除：先增加一个单独的节点：

node\_x = Node('英雄',name ='韦一笑')

test\_graph.create(node\_x)

test\_graph.run('match (n:英雄{name:\'韦一笑\'}) delete n')

# 删除一个节点及与之相连的关系

test\_graph.run('match (n:英雄{name:\'韦一笑\'}) detach delete n')

# 删除某一类型的关系

test\_graph.run('match ()-[r:喜欢]->() delete r;')

# 删除子图

# delete(self, subgraph)

**改**

改的基础也是查询，查到就可以改，因此本文的重点放在查询上，下面示例简单修改。

# 改

# 将狮王的武力值改为100

node\_9['武力值']=100

# 本地修改完，要push到服务器上哦

test\_graph.push(node\_9)

**查**

最重要的环节到了，一般来说，我们主要是查询，按照路径等、所有节点、关系等，我自己的学习需求还包括了一些统计特性的计算。主要使用下面的方式。  
为了使用更复杂查询，将图数据库扩充如下：

# 为了便于查询更多类容，新增一些关系和节点

transaction\_2 = test\_graph.begin()

node\_100 = Node('巾帼',name ='赵敏')

re\_100 = Relationship(node\_1,'Love',node\_100)

node\_101 = Node('巾帼',name ='周芷若')

re\_101 = Relationship(node\_1,'knows',node\_101)

re\_101\_ = Relationship(node\_101,'hate',node\_100)

node\_102 = Node('巾帼',name ='小昭')

re\_102 = Relationship(node\_1,'konws',node\_102)

node\_103 = Node('巾帼',name ='蛛儿')

re\_103 = Relationship(node\_103,'Love',node\_1)

transaction\_2.create(node\_100)

transaction\_2.create(re\_100)

transaction\_2.create(node\_101)

transaction\_2.create(re\_101)

transaction\_2.create(re\_101\_)

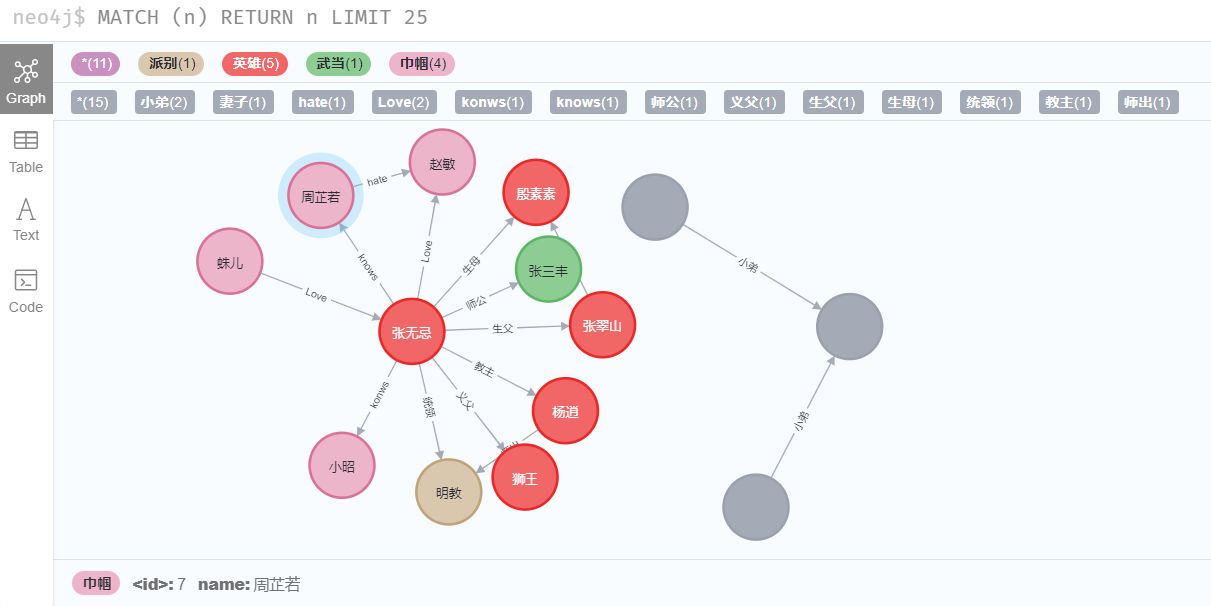
transaction\_2.create(node\_102)

transaction\_2.create(re\_102)

transaction\_2.create(node\_103)

transaction\_2.create(re\_103)

transaction\_2.commit()



1. NodeMatcher：定位满足特定条件的节点，基本语法：NodeMatcher（graph）.match(\*labels, \*\*properties)，结合不同的方法可以取得不同的效果。**主要**方法表如下所示：

|  |  |
| --- | --- |
| 方法名 | 功能 |
| first() | 返回查询结果第一个Node，没有则返回空 |
| all() | 返回所有节点 |
| where(condition,properties) | 对查询结果二次过滤 |
| order\_by | 排序 |

# 定义查询

nodes = NodeMatcher(test\_graph)

# 单个节点，按照label和name查询

## 查询节点：杨逍

node\_single = nodes.match("英雄", name="杨逍").first()

print('单节点查询：\n', node\_single)

## 按照label查询所有节点

node\_hero = nodes.match("英雄").all()

print('查询结果的数据类型:',type(node\_hero))

# 在查询结果中循环取值，用first()取出第一个值

i = 0

for node in node\_hero:

print('label查询第{}个为：{}'.format(i,node))

i+=1

## 按照name查询所有节点：用all()取出所有值

node\_name = nodes.match(name='张无忌').all()

print('name查询结果：',node\_name)

# get（）方法按照id查询节点

node\_id = nodes.get(1)

print('id查询结果：',node\_id)

# 运行结果

单节点查询：

(\_1:英雄 {name: '\u6768\u900d', 武力值: '100'})

查询结果的数据类型: <class 'list'>

label查询第0个为：(\_0:英雄 {name: '\u5f20\u65e0\u5fcc'})

label查询第1个为：(\_1:英雄 {name: '\u6768\u900d', 武力值: '100'})

label查询第2个为：(\_6:英雄 {name: '\u5f20\u7fe0\u5c71'})

label查询第3个为：(\_7:英雄 {name: '\u72ee\u738b'})

label查询第4个为：(\_8:英雄 {name: '\u6bb7\u7d20\u7d20'})

name查询结果： [Node('英雄', name='张无忌')]

id查询结果： (\_1:英雄 {name: '\u6768\u900d', 武力值: '100'})

2. NodeMatch：基本用法，NodeMatch(graph, labels=frozenset({}), predicates=(), order\_by=(), skip=None, limit=None)，可以看出，NodeMatcher和NodeMatch的区别吗？NodeMatch的参数和NodeMatcher的参数完全不同。后面是可以加很多条件的，包含的**主要**方法如下表：

|  |  |
| --- | --- |
| iter(match) | 遍历所匹配节点 |
| len(match) | 返回匹配到的节点个数 |
| all() | 返回所有节点 |
| count（） | 返回节点计数，评估所选择的节点 |
| limit(amount) | 返回节点的最大个数 |
| order\_by（\*fields） | 按指定的字段或字段表达式排序。要引用字段或字段表达式中的当前节点，请使用下划线字符 |
| where(\*predicates, \*\*properties) | 二次过滤 |

from py2neo import NodeMatch

nodess = NodeMatch(test\_graph,labels=frozenset({'英雄'}))

# 遍历查询到的节点

print('='\*15,'遍历所有节点','='\*15)

for node in iter(nodess):

print(node)

# 查询结果计数

print('='\*15,'查询结果计数','='\*15)

print(nodess.count())

# 按照武力值排序查询结果:注意引用字段的方式，前面要加下划线和点：\_.武力值

print('='\*10,'按照武力值排序查询结果','='\*10)

wu = nodess.order\_by('\_.武力值')

for i in wu:

print(i)

# 运行结果

=============== 遍历所有节点 ===============

(\_10:英雄 {name: '\u72ee\u738b', 武力值: 100})

(\_11:英雄 {name: '\u6bb7\u7d20\u7d20'})

(\_17:英雄 {name: '\u5f20\u65e0\u5fcc'})

(\_18:英雄 {name: '\u6768\u900d', 武力值: '100'})

(\_23:英雄 {name: '\u5f20\u7fe0\u5c71'})

=============== 查询结果计数 ===============

5

========== 按照武力值排序查询结果 ==========

(\_18:英雄 {name: '\u6768\u900d', 武力值: '100'})

(\_10:英雄 {name: '\u72ee\u738b', 武力值: 100})

(\_11:英雄 {name: '\u6bb7\u7d20\u7d20'})

(\_17:英雄 {name: '\u5f20\u65e0\u5fcc'})

(\_23:英雄 {name: '\u5f20\u7fe0\u5c71'})

3. RelationshipMatcher:用于选择满足一组特定标准的关系的匹配器。基础语法：relation = RelationshipMatcher(test\_graph)。

from py2neo import RelationshipMatcher

# 查询某条关系

relation = RelationshipMatcher(test\_graph)

# None表示any node哦！不是表示空

print('='\*10,'hate关系查询结果','='\*10)

x = relation.match(nodes=None, r\_type='hate')

for x\_ in x:

print(x\_)

# 增加俩关系

re1\_1 = Relationship(node\_101,'情敌',node\_102)

re1\_2 = Relationship(node\_102,'情敌',node\_103)

test\_graph.create(re1\_1)

test\_graph.create(re1\_2)

# 情敌查询结果

print('='\*10,'hate关系查询结果','='\*10)

x = relation.match(nodes=None, r\_type='情敌')

for x\_ in x:

print(x\_)

# 运行结果

========== hate关系查询结果 ==========

(周芷若)-[:hate {}]->(赵敏)

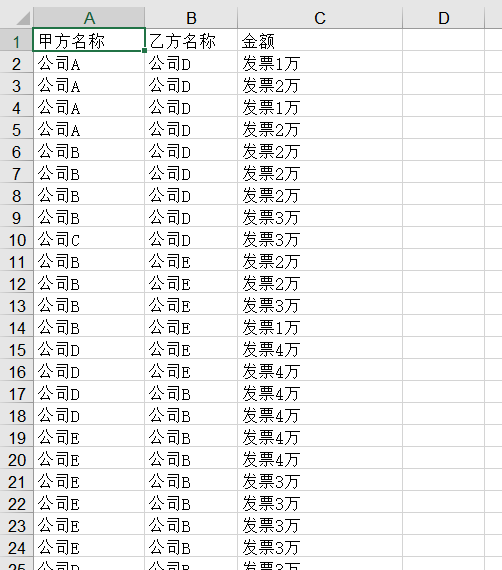
========== hate关系查询结果 ==========

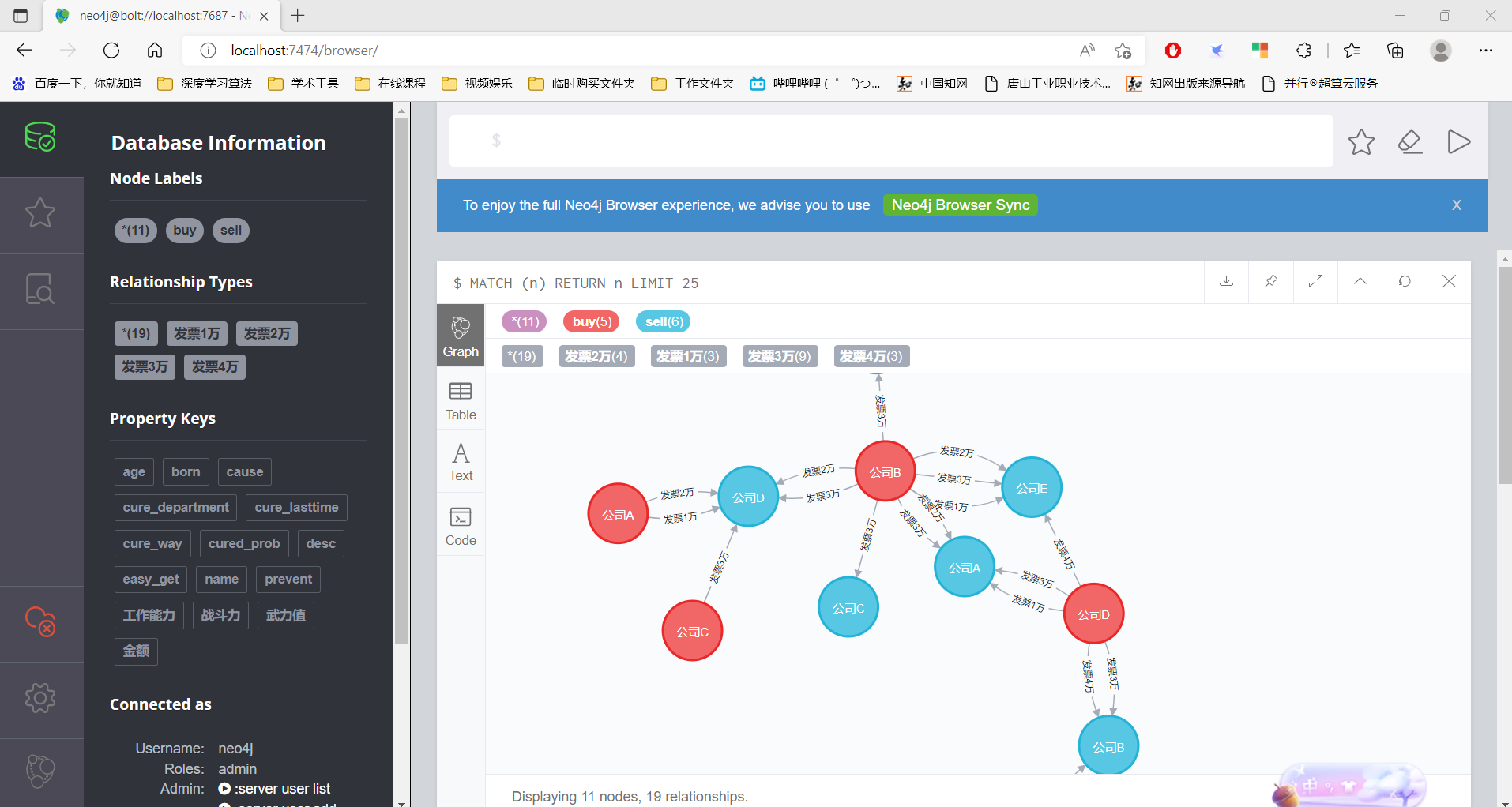
(周芷若)-[:情敌 {}]->(小昭)

(小昭)-[:情敌 {}]->(蛛儿)

4. RelationshipMatch:基本语法：RelationshipMatch(graph, nodes=None, r\_type=None, predicates=(), order\_by=(), skip=None, limit=None)。理解可以按照NodeMatch，不多作介绍了~

# 04 建立基础图数据库





# 05 医疗知识图谱问答机器人

中科院软件所刘焕勇老师在[github](https://so.csdn.net/so/search?q=github&spm=1001.2101.3001.7020)上的开源项目，基于知识图谱的医药领域问答项目

<https://github.com/liuhuanyong/QASystemOnMedicalKG>

配套学习网址

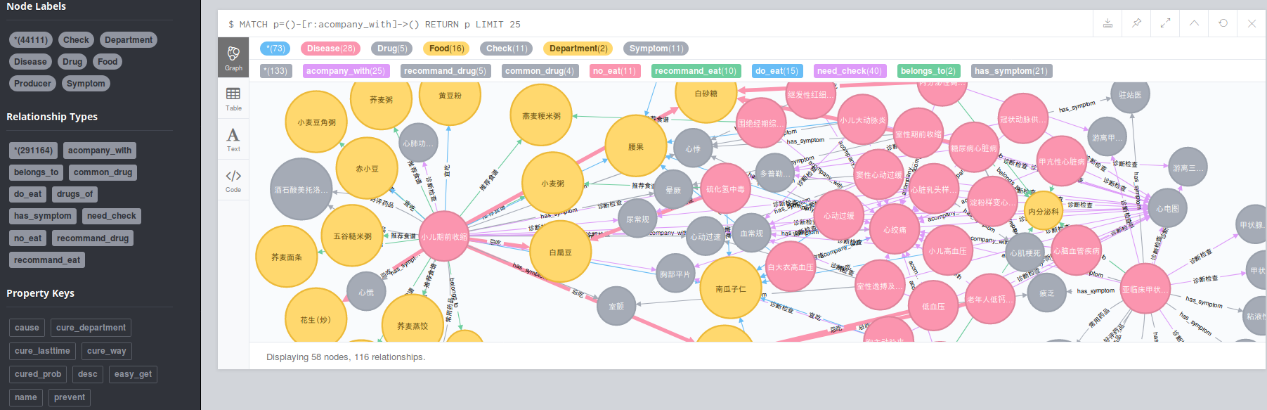
[医药领域知识图谱快速及医药问答项目（项目全过程）\_奔跑的蜗牛2.0的博客-CSDN博客](https://blog.csdn.net/qq_52062182/article/details/124542279)

[医药领域知识图谱快速及医药问答项目\_liuhuanyong\_iscas的博客-CSDN博客\_医药知识图谱](https://blog.csdn.net/lhy2014/article/details/82953792?spm=1001.2101.3001.6650.3&utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7Edefault-3-82953792-blog-124542279.pc_relevant_default&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7Edefault-3-82953792-blog-124542279.pc_relevant_default&utm_relevant_index=5)

[基于医疗知识图谱的问答系统源码详解\_vivian\_ll的博客-CSDN博客\_基于医疗知识图谱的问答系统](https://blog.csdn.net/vivian_ll/article/details/89840281?spm=1001.2101.3001.6650.2&utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7Edefault-2-89840281-blog-124542279.pc_relevant_default&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7Edefault-2-89840281-blog-124542279.pc_relevant_default&utm_relevant_index=4)

[基于医疗知识图谱的问答实践中遇到的问题\_vivian\_ll的博客-CSDN博客](https://blog.csdn.net/vivian_ll/article/details/89337931)

运行效果图：



项目运行方式

1、配置要求：要求配置neo4j数据库及相应的python依赖包。neo4j数据库用户名密码记住，并修改相应文件。

2、知识图谱数据导入：python build\_medicalgraph.py，导入的数据较多，估计需要几个小时。

3、启动问答：python chat\_graph.py



1.2 刘焕勇老师源程序脚本目录

prepare\_data/datasoider.py：网络资讯采集脚本

prepare\_data/max\_cut.py：基于词典的最大向前/向后切分脚本

build\_medicalgraph.py：知识图谱入库脚本

├── README.md

├── \_\_pycache\_\_ \\编译结果保存目录

│ ├── answer\_search.cpython-36.pyc

│ ├── question\_classifier.cpython-36.pyc

│ └── question\_parser.cpython-36.pyc

├── answer\_search.py

├── answer\_search.pyc

├── build\_medicalgraph.py \\知识图谱数据入库脚本

├── chatbot\_graph.py \\问答程序脚本

├── data

│ └── medicaln.json \\本项目的全部数据，通过build\_medicalgraph.py导neo4j

├── dict

│ ├── check.txt \\诊断检查项目实体库

│ ├── deny.txt \\否定词库

│ ├── department.txt \\医疗科目实体库

│ ├── disease.txt \\疾病实体库

│ ├── drug.txt \\药品实体库

│ ├── food.txt \\食物实体库

│ ├── producer.txt \\在售药品库

│ └── symptom.txt \\疾病症状实体库

├── document

│ ├── chat1.png \\系统运行问答截图01

│ ├── chat2.png \\系统运行问答截图01

│ ├── kg\_route.png \\知识图谱构建框架

│ ├── qa\_route.png \\问答系统框架图

├── img \\README.md中的所用图片

│ ├── chat1.png

│ ├── chat2.png

│ ├── graph\_summary.png

│ ├── kg\_route.png

│ └── qa\_route.png

├── prepare\_data

│ ├── build\_data.py \\数据库操作脚本

│ ├── data\_spider.py \\网络资讯采集脚本

│ └── max\_cut.py \\基于词典的最大向前/向后脚本

├── question\_classifier.py \\问句类型分类脚本

├── question\_classifier.pyc

├── question\_parser.py \\问句解析脚本

├── question\_parser.pyc

1.3 医药领域知识图谱规模

1.3.1 neo4j图数据库存储规模

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-VF6K2vDa-1620385989178)(https://github.com/liuhuanyong/QABasedOnMedicalKnowledgeGraph/blob/master/img/graph\_summary.png)]

1.3.2 知识图谱实体类型

实体类型 中文含义 实体数量 举例

Check 诊断检查项目 3,353 支气管造影;关节镜检查

Department 医疗科目 54 整形美容科;烧伤科

Disease 疾病 8,807 血栓闭塞性脉管炎;胸降主动脉动脉瘤

Drug 药品 3,828 京万红痔疮膏;布林佐胺滴眼液

Food 食物 4,870 番茄冲菜牛肉丸汤;竹笋炖羊肉

Producer 在售药品 17,201 通药制药青霉素V钾片;青阳醋酸地塞米松片

Symptom 疾病症状 5,998 乳腺组织肥厚;脑实质深部出血

Total 总计 44,111 约4.4万实体量级

1.3.3 知识图谱实体关系类型

实体关系类型 中文含义 关系数量 举例

belongs\_to 属于 8,844 <妇科,属于,妇产科>

common\_drug 疾病常用药品 14,649 <阳强,常用,甲磺酸酚妥拉明分散片>

do\_eat 疾病宜吃食物 22,238 <胸椎骨折,宜吃,黑鱼>

drugs\_of 药品在售药品 17,315 <青霉素V钾片,在售,通药制药青霉素V钾片>

need\_check 疾病所需检查 39,422 <单侧肺气肿,所需检查,支气管造影>

no\_eat 疾病忌吃食物 22,247 <唇病,忌吃,杏仁>

recommand\_drug 疾病推荐药品 59,467 <混合痔,推荐用药,京万红痔疮膏>

recommand\_eat 疾病推荐食谱 40,221 <鞘膜积液,推荐食谱,番茄冲菜牛肉丸汤>

has\_symptom 疾病症状 5,998 <早期乳腺癌,疾病症状,乳腺组织肥厚>

acompany\_with 疾病并发疾病 12,029 <下肢交通静脉瓣膜关闭不全,并发疾病,血栓闭塞性脉管炎>

Total 总计 294,149 约30万关系量级

1.3.4 知识图谱属性类型

属性类型 中文含义 举例

name 疾病名称 喘息样支气管炎

desc 疾病简介 又称哮喘性支气管炎…

cause 疾病病因 常见的有合胞病毒等…

prevent 预防措施 注意家族与患儿自身过敏史…

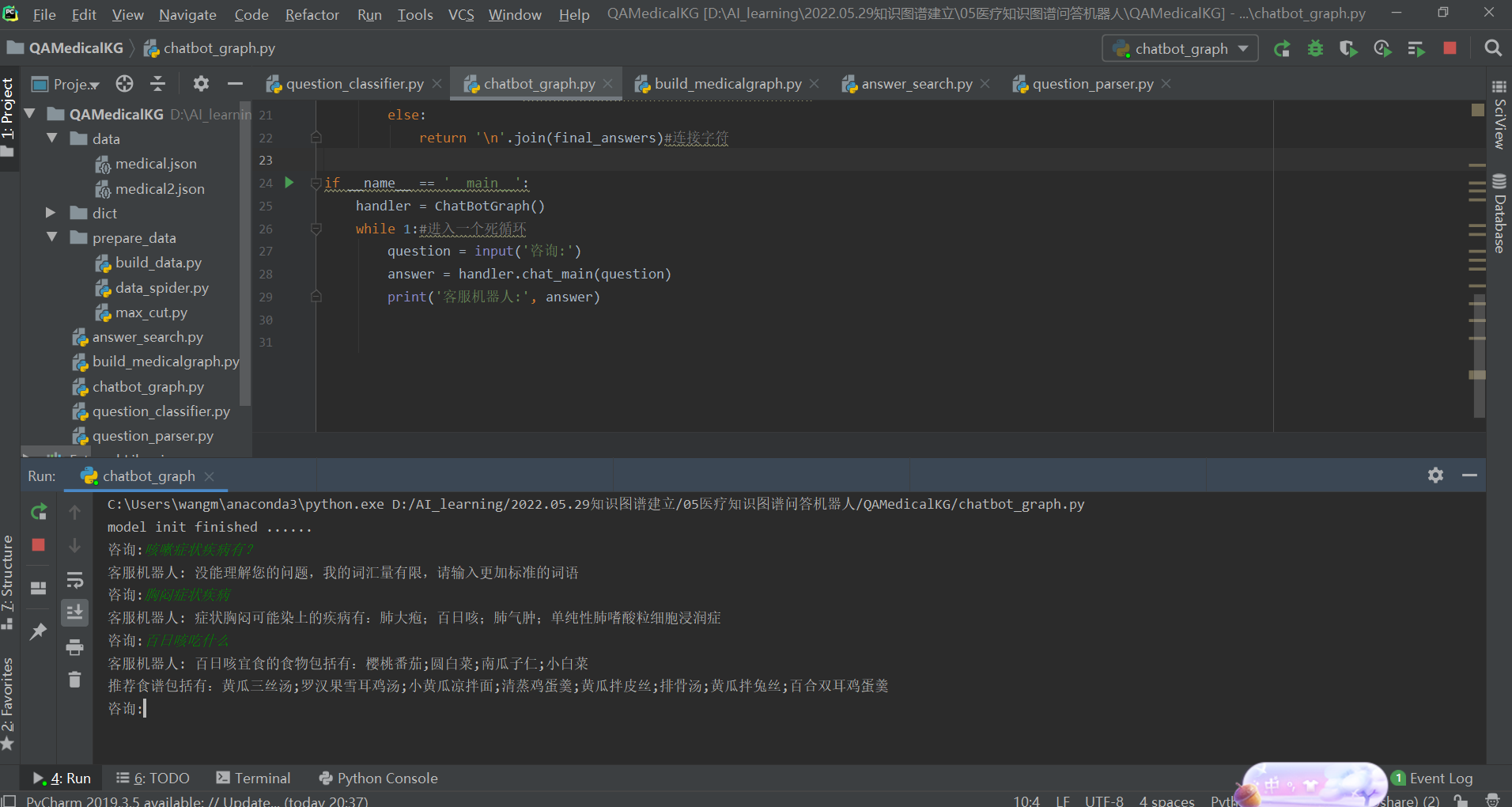
cure\_lasttime 治疗周期 6-12个月

cure\_way 治疗方式 “药物治疗”,“支持性治疗”

cured\_prob 治愈概率 95%

easy\_get 疾病易感人群 无特定的人群

输入一个症状等问句，去图数据库各个节点去找



对节点进行解析，首先定位到百日咳这个节点，再通过吃什么这个关系去找。

比如我描述一下自己的症状，这个智能机器人就会告诉我可能有什么症状，去什么科室挂号，做什么检查，检查花多少钱，避免社恐的自己去很多人的咨询点问人。

知识图谱跟业务更相关而不是技术，考虑要解决什么问题，有目的性的选择数据，搭建结构

import ahocorasick如果报错，可以百度搜索如何安装这个包

pip3 install pyahocorasick -i https://pypi.tuna.tsinghua.edu.cn/simple/

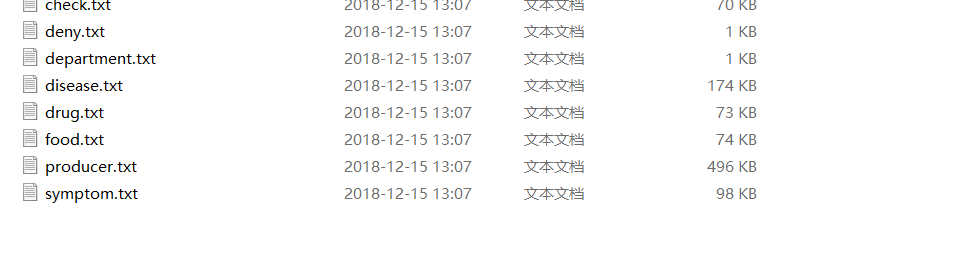
[(1条消息) ahocorasick安装\_wavehaha的博客-CSDN博客\_ahocorasick安装](https://blog.csdn.net/wavehaha/article/details/124705591)

核心：通过NLP的方法对语义进行识别，提取关键词

通过少量数据进入debug模式，了解建立过程

比如咨询输入：百日咳吃点什么，会分割出百日咳和吃 两个关键词，再寻找合适的关系进行匹配。百日咳可能有很多类型关系，寻找和吃最匹配的

与这里存储的单词进行匹配



图谱的意义不仅仅是得到一个漂亮的可视化，而是得到一个有智能分析的工具，快速返回想要的信息